Documentation for Pose Estimation

Last modified 6 May 2012 by Aaron Hallquist.

This document outlines the modular pose estimation which is attached to the query image retrieval system. Inputs, outputs, and important functions are described in detail. For a high level description of the algorithm, see the ACM conference submission in this directory. For more detailed questions, email aaron.hallquist@gmail.com.

CONTENTS

1. System Outline

Pose estimation is called as a module of image retrieval, with its run parameters set in the Context object as described in the Query System Documentation by Eric Liang. To run in conjunction with retrieval, set C.solve_pose=True and call system.characterize(C). An example of this using the Oakland dataset is [queryOakland.py].

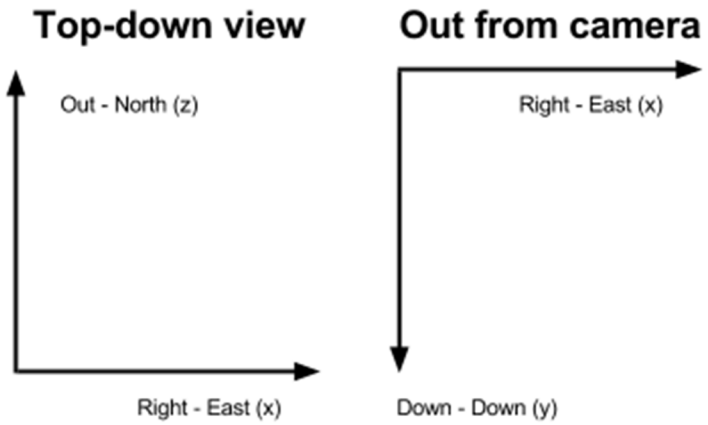
[computePose.py] serves as the highest level collection of functions which implements pose estimation. Within this file, the top level function is estimate_pose(), which is called from system.match(). This function compiles all necessary information for determining the query's pose, and calls into the functions which perform vanishing point alignment and homography estimation. It also outputs the pose estimation results into files for analysis in Matlab. Comparisons to ground truth are made, requiring a ground truth file.

[vp_analysis.py] contains all functions related to finding the vanishing points of the query and database image. Its top level function is getQNyaws(), called from computePose.estimate_pose(), detects lines in the query and database and assembles them into vanishing points, attempting to find common building faces between the images by aligning the vanishing points in the world frame. The output of this function is a list of aligned vanishing points, the "yaw direction" of the building faces they represent, and an estimate for the cell phone yaw orientation.

[solveHomography.py] contains all functions related to estimating a homography transformation between the database and query images. This file requires as input the set of feature correspondences. The top level function is constrainedHomography(), which runs a RANSAC loop based on different run parameters and outputs the position and orientation estimate as well as the set of inliers and other information about the run.

2. Coordinate System

The coordinate system is a right-handed system which is set up to transfer well to and from the image from to the world frame. To do this, we consider the following setup. If the camera is facing directly north, perpendicular to gravity, and is in a horizontal arrangement, then we consider it to have yaw, pitch, and roll of 0 degrees. Then we consider such a camera, where we can represent the coordinate system for both the camera and the world on top of each other. This is shown below in a top-down view (e.g. plane to ground type view) and a view out from this camera (facing directly north view).



That is, the coordinate x represents longitudinal change (east-west) in the world frame and represents pixel change to the right in the camera frame. The coordinate y represents altitude change (up-down) in the world frame and pixel change down in the camera frame. The coordinate z represents latitudinal change (north-south) in the world frame and the direction out from the camera in the camera frame.

3. Inputs

- All image retrieval inputs outlined in the Query System Documentation.
- CONTEXT VARIABLES
    - C.solve_pose : Boolean which runs pose estimation if set to True.
    - C.hiresdir : Directory containing the database high resolution images.
    - C.pose_remove : List of input query names which will not be evaluated by pose estimation.
    - C.pose_param : Dictionary of various run parameters for pose estimation (see Section 2.1).

2.1) C.pose_param

C.pose_param is a python dictionary containing many of the run parameters for pose estimation. The elements of the dictionary include:

resultsdir	Directory where output results are saved.
run_info	Output file listing some run parameters.
pose_file	Output file containing main results of the query pose estimation. See computePose.estimate_pose() for format.
extras_file	Output file containing extra information about the query run. See computePose.estimate_pose() for format.
cheat	Boolean deciding whether ground truth yaw is used or if it is computed from vanishing points.
runflag	Sets run condition for homography estimation. Only runflag=11 is guaranteed to work properly at this point: 0-7) Directly calls constrainedHomography() with the runflag given. See Section 4.3 on solveHomography.constrainedHomography(). 8-9) Deprecated. 10) Current approach solving for scale outside [solveHomography.py]. Calls solveNorm() inside [computePose.py] which calls constrainedHomography() with runflag=3. 11) Current approach solving for scale inside [solveHomography.py]. Calls solveNorm() inside [computePose.py] which calls constrainedHomography() with runflag=7.
remove_ground	Boolean which removes the database ground features from homography estimation if set to True. Requires plane data in the database.
use_weight	Boolean which uses a depth weighting for each feature correspondence. See computePose.estimate_pose() for implementation.
solve_bad	Boolean which runs pose estimation for *every* query in the set if set to True.
maxratio	Parameter for ratio test. See computePose.highresSift () for use.
ransac_iter	Maximum number of RANSAC iterations run for each query.
inlier_error	Base error threshold used for inlier set of RANSAC loop.

3. Outputs

All outputs are compiled into files stored in the directory C.pose_param['resultsdir']. There are six output file types:

1) ***run_info.txt*** : Contains information about the run parameters.
2) ***pose_results.txt*** : Contains the main results for each query, with each line in the text file outputting information for that query's results. Formatted for reading by Matlab functions described in Section 6.
3) ***extras.txt*** : Contains additional information about the query results. Again is formatted for reading by Matlab functions described in Section 6.
4) ***data_[[query]].txt*** : Contains information about an individual query's run, formatted for reading by humans. There is one of these files for each query.
5) ***[[query].jpg*** : Shows an image of the query and database image with feature matches. Its filename gives some indicators for how the query performed, and can be filtered with these. There is one of these images per query.

6) ***matches_[[query]].pkl*** : Python dictionary saved using the pickle package. The dictionary contains information on the output feature matches and is just the internal dictionary *matches* saved on disk; it can be read by loading it with the pickle package. There is one file per query.

3.1) File format for data_[[query]].txt:

This file will be in the following format, where X | Y | Z : a | b | c means X = a, Y = b, Z = c. Specifics are explained following each line, using a comment marker //.

```
Number of matches | Number of queries | ratio: 1431 | 1431 | 1.00
// Number of queries is the number of query SIFT features matched
// to a database feature. Number of matches is the total number
// of matched feature pairs. Ratio divides the two.

Plane Yaw | DB Plane | Confidence | Error = 201 | 0 | 1.00 | 6
// There is one line here for each plane detected from vanishing
// points and from the database. Plane Yaw is the yaw direction
// of the plane normal. DB Plane is the index for which database
// plane it refers to, where 0 = not a database plane. Confidence
// is a confidence parameter for the plane, largely unused. Error
// is a guess at the plane error based on the ground truth plane
// yaw reported, but should not be taken as an indicator, as
// there can be more than one plane.

VP Yaw | Confidence | Error : 47 | 0.33 | 5
// VP Yaw is the yaw orientation of the cell phone as solved by
// vanishing point alignment. Confidence is a parameter largely
// unused now. Error is the error of this estimated yaw with
// respect to the ground truth yaw.

Cell yaw | True yaw | Plane : 30 | 52 | 207
// This line reports the ground truth parameters, including the
// reported cell phone yaw (generally inaccurate), the ground
// truth yaw from Google Earth, and the ground truth plane.

-------------------------------------------------

VP yaw | Computed yaw | Actual yaw | Error : 47 | 47 | 52 | 5
// This reports the vanishing point yaw again, what yaw was used
// or computed by the homography estimation, the ground truth
// yaw, and the error between the computed yaw and ground truth.

Computed Normal | Actual Normal | Error : 201 | 207 | 6
// This reports the normal used or computed in homography
// estimation, the actual ground truth value, and the error.
// Actual value may be off if computation uses different plane.

Translation (x|y|z): -3.5 | 3.4 | -14.8
// This is the translation computed from homography estimation,
// in world coordinates where x = East, y = Down, z = North.
```

```
True position (x|-|z) : -3.0 | - | -12.6
// This is the true translation as measured with the ground
// truth. The y translation is unknown.

Angle error | Location error : 0 | 2.2
// This reports the error in translation in both angle between
// true translation and computed in degrees, as well as pure
// distance error in meters.

Number of Inliers | Total Matches | Ratio : 52 | 1124 | 0.05
// Total matches is how many feature pairs were used as input to
// homography estimation. Number of inliers is how many were
// considered inliers, while Ratio is the fraction.

Reprojection error | Homography confidence : 0.005 | 8.5
// This reports the average reprojection error over all inliers
// and the confidence of the solution based on the inliers.

Valid Homographies | Iterations | Ratio : 205 | 18350 | 0.011
// This reports how many solutions passed the validity checks in
// the ransac loop, the total number of solutions (iterations),
// and the ratio of the two.
```

4. High Level Functions and Variables

4.1) Function in [computePose.py]:

estimate_pose(C , Q , dbmatch , gtStatus)

This function compiles all necessary information for and performs pose estimation. It has outputs, but they are not currently used and are not important. The important outputs are saved to disk. The inputs are listed above and are defined as:

- **C:** The Context object for the query system and pose estimation.
- **Q:** The Query object containing information and functions relating to a specific query.
- **dbmatch**: The name id for the matching database image obtained by image retrieval.
- **gtStatus:** Obsolete.

4.2) Function in [vp_analysis.py]:

vyaw, vnorms = getQNyaws(C , Q , qimg , dimg , qsource , tyaw)

This function detects image vanishing points and estimates the yaw orientation of the cell phone. Its inputs are defined as:

- **C:** The Context object for the query system and pose estimation.
- **Q:** The Query object containing information and functions relating to a specific query.
- **qimg:** The full path for the high resolution jpg of the query image.

- **dimg:** The full path for the high resolution jpg of the database image.
- **qsource:** The query image object containing information for the cell phone image.
- **tyaw:** The ground truth yaw, which is set only if C.pose_param['cheat']=True; if False, tyaw is set to NaN and ignored inside this function.

The outputs are defined as:

- **vyaw:** The yaw orientation estimate from the vanishing point algorithm.
- **vnorms:** A list of the directional yaws for every detected plane in the scene.

4.3) Function in [solveHomography.py]:

```
matches, pose = constrainedHomography(
matches , wRd , wRq , qYaw , nYaw , runflag , maxerr , maxiter , minI , yrestrict )
```

This function estimates the homography transformation from the database image to the query image, using feature correspondences and various constraints known. The inputs are:

- **matches:** Dictionary of information about feature correspondences. See Section 4.5.
- **wRd:** Rotation matrix describing the orientation of the database image in the world.
- **wRq:** Rotation matrix describing the orientation of the query image in the world. The query yaw may use yaw from cell phone or from vanishing points, based on context.
- **qYaw:** The yaw orientation of the cell phone. Set to NaN if unknown.
- **nYaw:** The yaw of a known plane in the scene. Set to NaN if unknown.
- **runflag:** An integer from 0 to 7 representing 3 binary (2^3) run parameters. The definition for each value can be found in the code for this function. Generally **runflag** is set to 7, representing **qYaw** known, **nYaw** known, and scale solved inside the RANSAC loop.
- **maxerr:** The error threshold for inliers in our RANSAC loop.
- **maxiter:** The maximum number of iterations we allow the RANSAC loop to run for.
- **minI:** The minimum number of inliers required to consider a solution valid.
- **yrestrict:** A boolean which, if set True, places additional restrictions to declare a solution valid. See code for details on these restrictions.

The outputs are **matches** and **pose**, which are defined in Sections 4.5 and 4.6, respectively.

4.4) Function in [solveEssMatrix.py]:

```
matches, pose = constrainedEssMatrix(
matches , wRd , wRq , qYaw , runflag , maxerr , maxiter )
```

This function estimates the essential matrix transformation from the database image to the query image, using feature correspondences and various constraints known. It is not as well developed as constrainedHomography(). The inputs are:

- **matches:** Dictionary of information about feature correspondences. See Section 4.5.
- **wRd:** Rotation matrix describing the orientation of the database image in the world.
- **wRq:** Rotation matrix describing the orientation of the query image in the world. The query yaw may use yaw from cell phone or from vanishing points, based on context.
- **qYaw:** The yaw orientation of the cell phone. Set to NaN if unknown.
- **runflag:** An integer from 0 to 3 representing 2 binary (2^2) run parameters. The definition for each value can be found in the code for this function.
- **maxerr:** The error threshold for inliers in our RANSAC loop.
- **maxiter:** The maximum number of iterations we allow the RANSAC loop to run for.

The outputs are defined as:

- **matches:** Dictionary of information about feature correspondences. Additional information is stored in the dictionary about the run, including the inlier set. The set of parameters present in this dictionary will be a subset of those defined in Section 4.5.
- **pose:** This array differs from the array defined in Section 4.6, as there is no scale factor nor is there a plane parameter. Therefore the first three elements represent the directional translation in the world frame and the last element represents the yaw orientation of the cell phone.

4.5) Dictionary in [computePose.py]:

matches

This dictionary contains the following information about feature correspondences and the homography estimation run conditions. This list is not complete, but covers important parameters. See the use in code for any not listed.

nmat	Number of feature correspondences in the dictionary.
numq	Number of query features in the dictionary. Obsolete now, as numq=nmat.
qidx	A mapping implicitly indicating which query features are duplicated. Obsolete now, as query features cannot be duplicated in the set of correspondences.
q2d	nmat x 2 array containing the (x,y) pixel values for each query feature.
qprm	nmat x 2 array containing the (size,grad) values for each query feature.
qray	nmat x 3 array containing the directional ray for the query feature, in the query coordinate frame, obtained by pre-multiplying by the inverse camera matrix.
d2d	nmat x 2 array containing the (x,y) pixel values for each database feature.
dprm	nmat x 2 array containing the (size,grad) values for each database feature.
dray	nmat x 3 array containing the directional ray for the db feature, in the db coordinate frame, obtained by pre-multiplying by the inverse camera matrix.
ddep	nmat length array, containing the depth of the database feature
w3d	nmat x 3 array containing the absolute 3d position of the database feature, in the world coordinate frame, obtained by multiplying by dray by ddep and rotating the resulting vector into the world frame.
nnd	nmat length array with the nearest neighbor distance for each correspondence.

plane	nmat length array containing the integer represented plane on which the database feature lies. More details on database planes in Section 8.
weight	nmat length array with the importance of a correspondence, based on ddep.
imask	nmat length boolean array defining the inlier set.
domplane	Integer defining the db plane which a homography estimation used.
ifrc	Fraction of all matches that are inliers = sum(imask)/nmat.
iconf	Confidence of inlier set, used to decide between homography estimates.
numi	Number of inliers in the inlier set = sum(imask).
runflag	Runflag parameter used in this homography estimation.
rperr	The average reprojection error of all inliers.
niter	Total number of iterations executed in RANSAC loop.

4.6) Array in [computePose.py]:

pose

This is a length-6 array representing the computed pose of the query image. pose[:3] defines the direction of translation from the database to query image in the world frame, with norm 1. pose[3] represents the yaw orientation of the query image, pose[4] represents the yaw direction of the scene's plane normal, and pose[5] represents the scale factor of the whole system, so that pose[5]*pose[:3] is the 3d translation.

5. Utility Functions

5.1) In [computePose.py]: highresSift()

This function does a nearest neighbor search over the query and database features to find feature correspondences. It returns the first instantiation of the dictionary **matches**, described in Section 4.5. This first instantiation is saved to disk in order to avoid the search in future runs.

5.2) In [computePose.py]: getGTpose()

This function reads the ground truth file, reading the query image latitude, longitude, yaw orientation, and directional yaw for the scene's plane normal. The ground truth file is required for the system to work and is described in more detail in Section 9.

5.3) In [computePose.py]: match_info()

This function removes ground features and adds depth information to the **matches** dictionary, among a few other things. This is the second instantiation of **matches**.

5.4) In [computePose.py]: solveNorm()

This function calls solveYaw(), described in Section 5.5. The inputs include a list of candidate plane yaws, in order to solve a homography estimate for each plane yaw. The best homography among all candidate planes is chosen based on the best inlier confidence. This function alters **matches** one last time, adding an inlier set and other homography estimation run information.

5.5) In [computePose.py]: solveYaw()

This function does the same as solveNorm(), except over a set of candidate yaw orientations rather than candidates planes. However, this generally is not really used anymore, as only one candidate yaw orientation is generally used.

5.6) In [vp_analysis.py]: callLSD()

This function does a terminal call to Matlab in order to extract lines from an image using the LSD algorithm. Even though the LSD implementation is in C, the Matlab wrapper already existed, so we used it rather than develop a Python wrapper for the C function. The lines are saved to disk to avoid this Matlab call in future runs. The path to Matlab's [call_lsd.m] must be correct for this to work.

5.7) The file [geom.py]

This file contains most of the geometric and algebraic utility functions, including transformations from yaw, pitch and roll to a rotation matrix and back, the conversion from latitude and longitude to (x,z) coordinate locations, and the retrieval of the camera matrix based on horizontal field of view and image size. Relating to many of these functions, the coordinate systems are described as follows.

Coordinate Systems

(x,y,z) in the world frame represent (east,down,north) translations. In this way yaw, pitch, and roll are rotations about the y, x, and z axes respectively. This coordinate system may seem strange, but it is motivated by camera imagery. In camera frames like the query or database camera, (x,y,z) represent (right,down,out) translations. This conforms to traditional camera coordinate systems, where x is the pixel motion to the right, y is the pixel motion down, and z is the direction out of the image which arises when using camera matrices. These definitions mean that when yaw, pitch, and roll are all 0, the cell phone camera directly facing north and perpendicular to gravity, held in the landscape orientation. Currently, the way we extract yaw, pitch, and roll from the camera using [android.py] and ImageoTag (see Query System Documentation) will only be valid for horizontally held cell phones. Modifications would need to be made to accommodate for portrait images.

6. Matlab Functions and Analysis

6.1) In /matlab/lsd: call_lsd.m

This function must be present to extract lines from an image. If present, the Python function callLSD() in [vp_analysis()] should correctly call this function, assuming the path present in callLSD() points to the directory containing [call_lsd.m].

6.2) In /matlab/pose: pose_analysis.m

This function reads the file [poseresults.txt] from the results directory. It then outputs the pose estimation performance for the query set in the form of graphs. The inputs are defined as:

- **runnum:** Run number, indicating which results set to analyze. Default = 0.
- **setnum:** The set number for the query set to analyze. Currently only supports two sets, 5 for the Berkeley set and 11 for the Oakland set. Default = 11.

- **gps:** A boolean which decides whether or not to plot the pose performance of GPS as well. Default = False.

Currently, past runs are tracked by adding a number to the end of the output directory before running the next instance of the same query set. For example, for set number 11, the Oakland dataset, the final couple of output directories are currently /ah/pose_runs/oakland (as shown in queryOakland.py). After the run is complete, you can immediately inspect the results by calling pose_analysis in the following manner

pose_analysis(0,11)

which reads the file [pose_results.txt] from the directory /ah/pose_runs/oakland. If you want to run another Oakland query with different parameters, you first must rename the final directory to something like /ah/pose_runs/oakland6. With this renaming, it is also recommended that you manually modify the file /ah/pose_runs/guide.txt to indicate what type of run is contained is oakland6. After renaming, you can now run a new query which will be stored in /ah/pose_runs/oakland and will not overwrite your previous run. You can inspect the performance of your old run by calling

pose_analysis(6,11)

Similary, if you have a run in /ah/pose_runs/berkeley12 from the Berkeley dataset (set number = 5), and you also want to compare these results to GPS, you can call pose_analysis in the following manner:

pose_analysis(12,5,1)

6.3) In /matlab/pose: prehom_analysis.m

This function is similar to [pose_analysis.m], but reads the file [extras.txt] and evaluates the performance of factors before homography estimation. In particular it evaluates the performance of yaw orientation estimates using vanishing points. The inputs are very similar to [pose_analysis.m]:

- **runnum:** Run number, indicating which results set to analyze. Default = 0.
- **setnum:** The set number for the query set to analyze. Currently only supports two sets, 5 for the Berkeley set and 11 for the Oakland set. Default = 11.

As an example, if you want to compare yaw orientation performance between your Berkeley set is /ah/pose_runs/berkeley7 with the compass yaw performance, you can call

prehom_analysis(7,5)

7. Python Dependencies

All Python requirements for image retrieval are also required for pose estimation, as it acts as a module. This includes all packages, as well as database and dataset formats.

Those packages specifically required by pose estimation and some notes on them are as follows:

numpy	Used for general math and random numbers

scipy	Used for optimization
pyflann	Used for nearest neighbor search. This probably will not work in the Windows environment, and is the primary reason our code sits in a Unix environment.
Image	Used for drawing output images and parsing dataset images.

In addition to packages, additional information about the database and dataset must be present above and beyond what is required by image retrieval. In particular, full resolution images for both the query and database images are used.

For the query dataset, this path is assumed to be relative to the path provided for low res images in image retrieval. That is, the high res images must be located in the sub-directory /hires. In this directory, the high res jpgs must be present, as well as high res pgm files and high res sift.txt feature files. The process of creating the latter two, .pgm and sift.txt files, is described in Section 9.

In contrast, the high resolution database directory is *not* relative to the low res directory, and contains more information. For each database image, you must have the id name with the following extensions:

- **.jpg :** The high resolution color image
- **.info :** An information file about the database image, similar to the low res .info file, but with different camera parameters (in particular the pixel width and height).
- **.pgm :** The high resolution B/W image used to extract SIFT features
- **sift.txt :** The extracted SIFT features for the database image.
- **-depth.png :** The high resolution depth map, containing a 16-bit unsigned integer at each pixel representing that pixels depth in cm (0 -> 65535). If the pixel value is 0 or 65535, then no depth is present. If in between, the units are cm, e.g. 12345 value represents a depth of 123.45 m.
- **-planes.png :** The high resolution plane map, containing an integer for each plane, so that a pixel value of X means that the pixel belongs to plane X. This file is optional, though if you do not have it, it is recommended that you set C.pose_param['remove_ground']=False (see Section 2.1).

If you are starting with panoramas and associated metadata and you need to extract these files, take a look at the file [extract_oakland.py]. It will definitely not work for a dataset you have, but contains the basic process, including the mapping from panorama to rectilinear images.

Finally, the pose estimation algorithm requires a hand constructed ground truth. However, if this is unnecessary, the file is still required for pose estimation to run, but will work with zeros for the ground truth values (Matlab analysis will not work as intended, however). The ground truth was done by visual inspection in Google Earth, and the format for the ground truth file is listed Section 9.

8. Matlab Dependencies

The Matlab function [call_lsd.m] requires all the necessary C functions for line extraction in the same directory. You can test that this is working by calling [compile.m] within the same directory.

As mentioned above, the Matlab analysis functions require a correct ground truth to function properly. In addition, the paths to result directories must be correct. These can be edited in the Matlab files.

9. Query and Database Setup

Creating .pgm and sift.txt files from .jpg

This is done with a Windows batch process. Look at the files stored in /scripts for examples.

Extracting rectilinear images and maps from panoramas

Nothing will do this specifically for you, unless what you have happens to be in a very special format. However, look at [extract_oakland.py] for the overall method to do this.

Ground truth creation and format

The pose estimation module requires query pose ground truth information. The format is in the following table, where each row represents a new line in the text file and each column is separated by a tab. The file must be in the low res query directory with the name gtLatLonNormYaw.txt

Query 1 name ID	Latitude	Longitude	Yaw of Plane	Yaw of Image
Query 2 name ID	Latitude	Longitude	Yaw of Plane	Yaw of Image
…	…	…	…	…
Query N name ID	Latitude	Longitude	Yaw of Plane	Yaw of Image

This ground truth file must exist, but does not have to have correct values. All zeros will work, as long as all query name IDs exist. Note that if the ground truth values are bad, the Matlab analysis functions will not produce meaningful results.

If you wish to ground truth, use Google Earth and visual inspection to estimate the position of the query image, its direction, and the direction of its dominant plane. The ruler tool will help to give you yaw of plane and yaw of image with its bearing value. The yaw values are in degrees, not radians.

Moving code to another computer

If you move the code and/or datasets to a different computer, keep in mind you must have all the dependent Python packages installed and the LSD Matlab code and its dependencies also must move. The following defined paths in the code must be updated:

- Path to [call_lsd.m] that is used by the function callLSD() in the file [vp_analysis.py].
- Path to the database images and other files must be updated (both the high and low res paths). This is modified in the outermost function, such as [queryOakland.py] or [querySystem.py].
- Path to the query set updated in [queryOakland.py].
- Path where the results directory is must be updated in ***both*** Python and Matlab: for Python this is C.pose_param['resultsdir'] in [queryOakland.py] and for Matlab this is in both the functions [prehom_analysis.m] and [pose_analysis.m]

10. Tutorial Function

A current, working version of the image retrieval and pose estimation module is on Gorgan, under /media/DATAPART1/oakland/app/dev-ah/project/src. In a terminal on Gorgan, cd to this directory and call the top level function with the following statement:

python queryOakland.py

The pose estimation results, as shown in [queryOakland.py], will be stored in /media/DATAPART2/ah/pose_runs/oakland. Note how C.selection can grab a subset of all queries. Comment that line out to test over all query images.

In addition, from Gorgan you can load up Matlab by opening a terminal and typing 'matlab'. When in there, change the directory to /media/DATAPART1/oakland/app/dev-ah/matlab/pose, where you can experiment with [pose_analysis.m] and [prehom_analysis.m]. Make the calls

pose_analysis(0,11,1)

prehom_analysis(0,11)

These calls will be more instructive with a full run, rather than one limited to only two query images.